Constraint Programming Based Large Neighbourhood Search for Energy Minimisation in Data Centres*

Hadrien Cambazard[1], Deepak Mehta[2], Barry O'Sullivan[2], and Helmut Simonis[2]

[1] G-SCOP, Université de Grenoble; Grenoble INP; UJF Grenoble 1; CNRS, France
hadrien.cambazard@grenoble-inp.fr
[2] INSIGHT Centre for Data Analytics
Department of Computer Science, University College Cork, Ireland
{d.mehta|b.osullivan|h.simonis}@4c.ucc.ie

Abstract. EnergeTIC is a recent industrial research project carried out in Grenoble on optimising energy consumption in data centres. We study the problem formulation proposed by EnergeTIC. The problem focuses on the allocation of virtual machines to servers with time-variable resource demands in data centres in order to minimise energy costs while ensuring service quality. We present a scalable constraint programming-based large neighbourhood search (CP-LNS) method to solving this challenging problem. We present empirical results that demonstrate that the industrial benchmarks can be solved to near optimality using our approach. Our CP-LNS method provides a fast and practical approach for finding high quality solutions for lowering electricity costs in data centres.

1 Introduction

Data centres are a critical and ubiquitous resource for providing infrastructure for banking, Internet and electronic commerce. They use enormous amounts of electricity, and this demand is expected to increase in the future. For example, a report by the *EU Stand-by Initiative* stated that in 2007 Western European data centres consumed 56 Tera-Watt Hours (TWh) of power, which is expected to almost double to 104 TWh per year by 2020.[3] Nevertheless, as reported by the consulting firm McKinsey, only 6-12% of electricity used by data centres can be attributed to the performance of productive computation [7]. Therefore, one of the optimisation challenges in the domain of data centres is to keep servers well utilised so that energy costs can be reduced.

Many data centres have the infrastructure in place for load migration. There are several reasons for migrating the load of one or more virtual applications from their current servers to different ones. For example, if the load on a server is very high, or if the server is about to shut down, then one might want to move some or all the virtual machines from that server to others. Also, if there is a server where the energy cost per unit of computation is cheaper, then one might want to reassign some virtual

* The authors want to acknowledge the industrial partners: Bull, Schneider Electric, Business & Decision and UXP as well as public research institutions: G2Elab, G-SCOP and LIG. The authors from UCC are supported by Science Foundation Ireland Grant No. 10/IN.1/I3032.

[3] http://re.jrc.ec.europa.eu/energyefficiency/html/standby_initiative_data_centers.htm

applications to that server so that the overall cost of energy consumption is reduced. In general, the challenge is to consolidate machine workload intelligently to ensure that servers are well utilised so that energy costs can be reduced.

In this paper we describe a constraint optimisation model for energy-cost aware data centre assignment systems which allocates virtual machines with time-variable demands to servers where the energy cost per unit of computation can vary between different locations. The problem we consider is defined by a set of servers and a set of virtual applications to be run on those servers over a given operating horizon. Each server is associated with a set of available resources, e.g. CPU, RAM, DISK etc. Each virtual application is associated with an optional initial server on which it is running, and a set of required resource values, which might be different over different time slots in our operating horizon. The solution of the problem is an assignment of virtual machines to servers at each time-period which respects a set of hard constraints. The objective is to take advantage of differences in energy costs across the servers, the requirements of virtual applications, the transition costs of switching the states of servers from ON to STANDBY and vice-versa, and by reassigning virtual applications to servers within a given data centre.

The remainder of the paper is organised as follows. First we describe the overall project and in particular energy and demand models to give the context of our work on solving the optimisation problem addressed in this paper. We then describe the formal definition of the problem before presenting a constraint optimisation formulation of it. The size of the instances of this problem can be prohibitively large for standard optimisation techniques, but needs to be solved quickly. Therefore, the challenge is to search for a good quality solution of a very large problem instance in a very limited timeframe. We present a constraint programming-based large neighbourhood search (CP-LNS) for solving this problem which scales significantly beyond commercial optimisation tools such as CPLEX.[4] The key idea behind CP-LNS is to repeatedly consider a sub-problem of the overall problem and re-optimise it using constraint programming. We present a systematic empirical evaluation of our CP-LNS approach. Empirical results obtained on real benchmarks demonstrate the scalability of our CP-LNS approach, and show that it provides a practical basis for solving this very important and challenging real-world problem.

2 Related Work

A variety of studies on allocating data and workload amongst multiple servers have been reported [9,5]. A mixed integer programming approach to dynamically configuring the consolidation of multiple services or applications in a virtualised server cluster has been proposed [11]. That work both focuses on power efficiency, and considers the costs of turning on/off the servers. However, it is assuming homogeneous workloads, e.g. web searches, where there is little uncertainty around the duration of tasks or the current cost of energy.

Constraint programming based approaches have also been used previously to solve some related problems [6,3,8]. In [3] a data centre is viewed as a dynamic bin packing

[4] `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

system where servers host applications with varying resource requirements and varying relative placement constraints. However, their work can be seen as a reactive approach where the servers are reconfigured when the current configuration is no longer viable. Therefore, their objective is to minimise the transition time for migrations of virtual machines, whereas we are concerned with minimising the energy consumption, and we plan the migrations for virtual machines in advance so that the configuration always remain viable.

A high-availability property for a virtual machine is defined in [8]. When a virtual machine is marked as k-resilient, as long as there are up to k server failures, it is guaranteed that it can be relocated to a non-failed host without relocating other virtual machines. The property of k-resiliency relies on static resource requirements of virtual machines whereas we are focusing on time variable resource demands.

Although in [6] an energy-aware framework is proposed for the reallocation of virtual machines in a data centre to reduce the power consumption, the goal is to find the best possible placement of virtual machines for a given time-period subject to service level agreements.

3 Energy and Demand Models

The problem studied in this paper comes from the EnergeTIC project which is accredited by the French government (FUI) [2].[5] EnergeTIC brought together four companies (Bull, Business & Decision Eolas, Schneider Electric, UXP) and several academic partners (G2Elab, G-SCOP, LIG). Its main objective is to control the energy consumption of a data centre and ensure that it is consistent with application needs, economic constraints and service level agreements. It focused on how to reduce energy cost by taking into account variable CPU requirements of the clients' applications, the wide range of IT equipment and virtualisation techniques. A tool was implemented and deployed in practice in a data centre designed by Eolas. The system developed by EnergeTIC is based on a model of the energy consumption of the various components in a data centre, a prediction system to forecast the demand and an optimisation component computing the placement of virtual machines onto servers. In the following we describe the energy and demand models of the system and in the remaining part of the paper we focus on the optimisation part.

3.1 Energy Model

Green data centres appeared as early as 2000 and focused on limiting the amount of energy that was not used for running the client's applications. The Power Usage Effectiveness (PUE) is a key indicator introduced by the Green Grid consortium [1] which measures the ratio between the total energy entering the data centre and the energy used by its IT systems (servers, networks, etc.). The power consumed by support equipment and infrastructure is regarded as an overhead according to this metric. A PUE value of 1

[5] Minalogic EnergeTIC is a global competitive cluster located in Grenoble France and fosters research-led innovation in intelligent miniaturised products and solutions for industry.

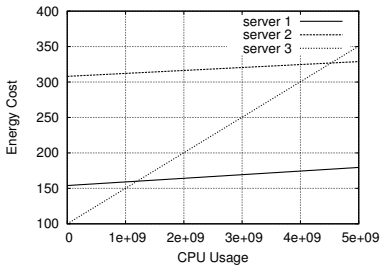

Fig. 1. Energy cost (Wh) vs CPU Usage (GHz) for 3 servers.

is a *perfect efficiency*. This indicator has been used to measure progress over the years. A value of 2.5 was common a few years ago whereas the current average in industry is around 1.7 with the most efficient data centres reaching 1.2 to 1.4.

The need to refine such metrics arose quickly, especially when considering that not all electrical power delivered by the IT equipment is transformed into value-adding computation. The Green Grid proposed a very fine-grained indicator called DCP (Data Center Productivity) for that purpose [1]. This metric, although very accurate, is not used in practice because of its complexity. The EnergeTIC project introduced two simple indicators related to usage efficiency. The first aims at checking the productive use of active resources while the second focuses on the energy consumed. This last energy indicator is defined as the ratio between the total energy consumed and the energy specifically used to run clients applications.

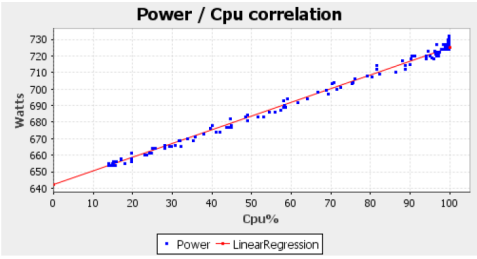

Fig. 2. Linear model of energy (courtesy to [2]).

The energy indicator relies on a model of the energy consumption of each piece of equipment, e.g. ventilation units, power supplies, heating/cooling systems, etc., as well as a wide range of IT equipment such as servers, storage, etc. The characterisation of the energy consumption of each piece of IT equipment was performed on a cooled rack provided by Bull which was instrumented with sensors. The rack contained a dozen of heterogeneous servers based on three types of processors: quadri, bi and mono. Different energy behaviour were used in various scenarios to perform the measurements. As an example, the energy cost of the power consumption of three different servers at

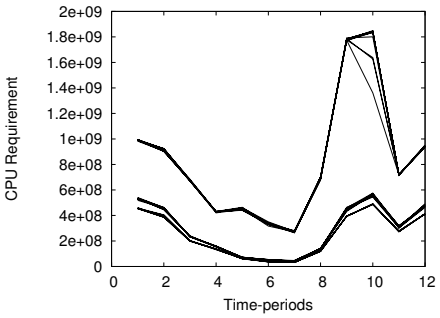

Fig. 3. Variable demands – Example 1.

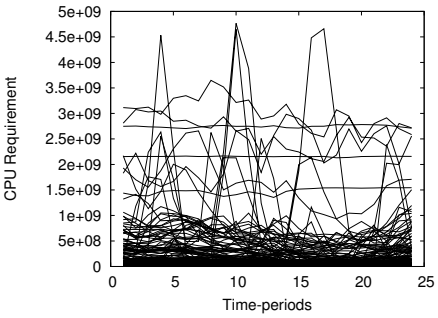

Fig. 4. Variable demands – Example 2.

different CPU loads taken from one of the problem instances is shown in Figure 1. Reality is often quite complex as performance also depends on other parameters such as the room temperature or the shared resources where contention can occur. However, a linear model was found accurate enough to model energy consumption of the servers. The model was computed by linear regression over the measures (see Figure 2). The measure of the CPU requirements of an application is more complex as it would need to be done on each possible server type. Therefore, in practice, a single measure was performed on a reference server [2].

3.2 Demand Model

The demands, i.e. the resource requirements, of the virtual machines in the benchmarks used in the experimental section originate from the Green Data Centre of Business & Decision Eolas located in Grenoble which started in 2011. It was used to study and validate the system operationally. It is a Tier IV centre instrumented with thousands of sensors spread over the site to monitor its energy consumption (IT, Security, monitoring, inverters, power supplies, etc.) and claims a PUE between 1.28 and 1.34. It deals with an heterogeneous demand: web applications, e-commerce, e-business, e-administration, etc. The data sets used to make an offline evaluation of the optimiser are obtained from historical data from this data centre. Two examples showing variable requirements of CPU usage (GHz) over 12 and 24 time-periods for multiple virtual machines taken from two problem instances is shown in Figures 3 and 4. An online evaluation of the optimiser was also performed in practice on a "sandbox" platform that reproduces the real environment with only three servers. Real applications were copied from the production environment to this restricted environment where the decisions proposed by the optimiser were implemented and evaluated.

4 Problem Description

We now describe the optimisation problem provided by EnergeTIC. The problem is to place a set of virtual machines on a set of servers over multiple time-periods in order

to minimise the energy cost of the data centre. The CPU usage of a virtual machine changes over time. At each time-period, we must ensure that the virtual machines have enough resources (CPU and memory). Let $V = \{v_1, \ldots, v_n\}$ be the set of virtual machines, $S = \{s_1, \ldots, s_m\}$ be the set of servers and $T = \{p_1, \ldots, p_h\}$ be the set of time-periods.

Virtual Machines. A virtual machine v_i is characterised by a memory consumption M_i independent of the time-period, a set $A_i \subseteq S$ of allowed servers where it can be hosted, and a potential initial server (for time-period p_0) denoted by $Iserv_i$ (which might be unknown). A virtual machine v_i has a CPU consumption U_{it} at time-period t.

Servers. A server s_j can be in two different states: ON=1 or STBY=0 (stand-by). It is characterised by: a CPU capacity $Umax_j$; a memory capacity $Mmax_j$; a fixed cost of usage $Emin_j$ (in Watt) when the server is ON; a unit cost τ_j per unit of CPU usage; a basic CPU consumption Ca_j when it is ON to run the operating system and other permanent tasks; an energy consumption $Esby_j$ when it is in state STBY; an energy consumption $Esta_j$ to change the state of the server from STBY to ON; an energy consumption $Esto_j$ to change the state of the server from ON to STBY; a maximum number $Nmax_j$ of virtual machines that can be allocated to it at any time-period; a set of periods $P_j \subseteq T$ during which s_j is forced to be ON; and a potential initial state $Istate_j \in \{0, 1\}$.

If a server is ON, its minimum cost is $Emin_j + \tau_j Ca_j$. Therefore, for the sake of simplicity, to compute the fixed energy cost of an active server we include the basic consumption Ca_j in $Emin_j$ and denote that by $Emin'_j = Emin_j + \tau_j Ca_j$. We also shift the CPU capacity of a server and denote that by $Umax'_j = Umax_j - Ca_j$.

Migrations. The maximum number of changes of servers among all virtual machines from one time-period to the next is denoted by N and the cost of a migration by $Cmig$.

This problem can be seen as a series of packing problems (one per time-period) in two dimensions (CPU and memory) that are coupled by the migration constraints and the cost for changing the state of a server. Figure 5 gives an overview of the problem. This example has four servers, each shown by a rectangle whose dimensions are representing the CPU and memory capacities of that server. A virtual machine is a small rectangle whose height (its CPU) varies from one period to the next. Therefore, the sum of the heights (CPU) must fit within the capacity (height of the rectangle), and similarly for the sum of the widths (memory) must fit within the available capacity (width of the rectangle). In this scenario, the CPU needs of some virtual machines decreases allowing us to find better packings and turn off two servers at $t + 1$.

5 Problem Formulation: Constraint Optimisation Model

We present the constraint optimisation model of the problem.

5.1 Variables

– Let $x_{it} \in A_i$ be the main integer decision variables that denote the server on which virtual machine v_i is running at time-period t. The constraint that a virtual machine

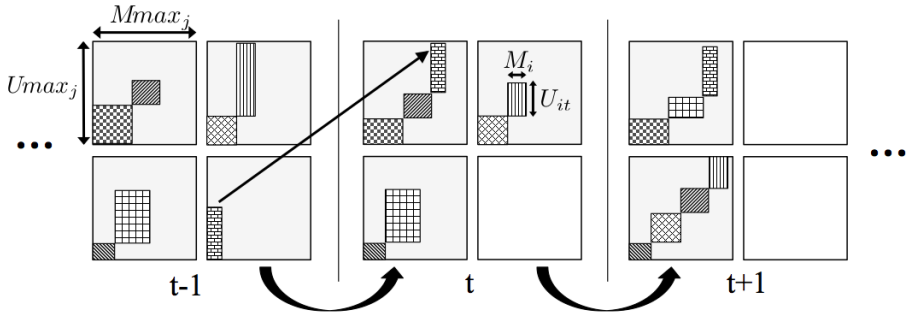

Fig. 5. A solution over three time-periods. Virtual machines migrate to turn off two servers at $t + 1$.

has to be on a server at any time and the forbidden servers for each machine are trivially enforced through the assignment of x to a value from its domain.

- Let $cpu_{jt} \in [0, Umax'_j]$ be the non-negative continuous variable that measures the CPU consumption of server s_j at period t.
- Let $mem_{jt} \in [0, Mmax_j]$ be the non-negative continuous variable that measures the memory consumption of server s_j at period t.
- Let $nvm_{jt} \in [0, Nmax_j]$ be an integer variable that denotes the number of virtual machines running on server s_j at time t.
- Let $cs_t \in [0, N]$ be an integer variable that denotes the number of virtual machines that change servers from time-period $t - 1$ to time-period t.
- Let $o_{jt} \in \{0, 1\}$ be a Boolean variable that is set to 1 if s_j is ON at time t, 0 otherwise.

The initial state is denoted by $t = 0$. For each server $s_j \in S$ and virtual machine $v_i \in V$ variables o_{j0} and x_{io} are also created.

5.2 Constraints

Capacity Constraint. The following constraints link the CPU and memory loads of a server to the virtual machines assigned to it.

$$\forall_{s_j \in S} \forall_{p_t \in T} : cpu_{jt} = \sum_{v_i \in V \wedge x_{it} = j} U_{it} \tag{1}$$

$$\forall_{s_j \in S} \forall_{p_t \in T} : mem_{jt} = \sum_{v_i \in V \wedge x_{it} = j} M_i \tag{2}$$

The constraint on the usage for CPU and memory on a server in any time-period must not exceed their capacities is trivially enforced through the upper bounds of the domains of cpu_{jt} and mem_{jt}, respectively.

Cardinality Constraint. The maximum number of virtual machines that can run on a server in any time-period is constrained:

$$\forall_{s_j \in S} \forall_{p_t \in T} : nvm_{jt} = |\{v_i | v_i \in V \wedge x_{it} = j\}| \tag{3}$$

Migration Constraint. The number of server changes over all virtual machines in any time-period is constrained:

$$\forall_{p_t \in T} : \quad cs_t = |\{v_i | v_i \in V \wedge x_{it-1} \neq x_{it}\}| \tag{4}$$

ON Constraint. A server is ON if it is hosting at least one virtual machine:

$$\forall_{v_i \in V} \forall_{p_t \in T} : \quad x_{it} = j \implies o_j = 1 \tag{5}$$

The following states the time-periods where a server has to be ON:

$$\forall_{s_j \in S} \forall_{p_t \in P_j} : \quad o_{jt} = 1 \tag{6}$$

When a server s_j is ON at two time-periods say t_a and t_c (where $t_c > t_a + 1$) it is better to leave it ON in between when it would cost more to switch it to STBY. The cost of putting s_j in ON state would be $(t_c - t_a) \times Emin'_j$ and the cost of putting s_j in STBY state would be $(t_c - t_a) \times Esby_j + Esta_j + Esto_j$. Thus, if $t_c - t_a < D$ where $D = \left\lceil \frac{Esta_j + Esto_j}{(Emin'_j - Esby_j)} \right\rceil$ then it is better to set $o_{jt_b} = 1$ for all t_b such that $t_c < t_b < t_a$. In other words in an **optimal** solution, any sequence of 0 values in the vector of variables $[o_{j1}, \ldots, o_{jh}]$ should be of length at least D. If not the cost can be improved by turning ON the corresponding server in the corresponding time-periods. This dominance rule can be encoded using the following set of constraints:

$$\forall_{t_a \in T} \forall_{t_a+1 < t_c \in T < t_a + D} \forall_{t_a < t_b < t_c} : \quad o_{jt_a} = 1 \wedge o_{jt_c} = 1 \implies o_{jt_b} = 1 \tag{7}$$

Similarly, for each server s_j we need to consider two special cases: the first time-period when the server s_j is ON and the last time-period when the server s_j is ON. If t_b is the first time-period when a server s_j is ON (where $t_b > 1$) it is better to leave it ON in all the time-periods before t_b if $t_b < D_f$ where $D_f = 1 + \left\lceil \frac{Esta_j}{(Emin'_j - Esby_j)} \right\rceil$.

$$\forall_{1 < t_b \in T < D_f} \forall_{1 \leq t_a < t_b} : \quad o_{jt_b} = 1 \implies o_{jt_a} = 1 \tag{8}$$

If t_a is the last time-period when a server s_j is ON (where $t_a < h$) it is better to leave it ON in all the time-periods after t_a if $t_a > D_l$ where $D_l = h - \left\lceil \frac{Esto_j}{(Emin'_j - Esby_j)} \right\rceil$.

$$\forall_{D_l < t_a \in T < h} \forall_{t_a \leq t_b \leq h} : \quad o_{jt_a} = 1 \implies o_{jt_b} = 1 \tag{9}$$

Initial State. If the initial configuration is given then the constraints $o_{j0} = Istate_j$ and $x_{i,0} = Iserv_i$ are enforced for each $s_j \in S$ and $v_i \in V$ respectively. Otherwise, the constraints $o_{j0} = o_{j1}$ and $x_{i0} = x_{i1}$ are enforced.

5.3 Objective Function

The objective is to minimise the sum of the following costs:

Migration Cost. The total migration cost is the total number of server changes over all virtual machines over all time-periods multiplied by the cost of migration:

$$Cmig\left(\sum_{t\in T} cs_t\right)$$

Transition Cost. The total transition cost is the sum of all the transitions of all servers from STBY state at time-period $t-1$ to ON state at time-period t, and vice-versa over all time-periods:

$$\sum_{s_j\in S}\left(\sum_{t\in T\wedge o_{jt-1}<o_{jt}} Esta_j + \sum_{t\in T\wedge o_{jt-1}>o_{jt}} Esto_j\right)$$

CPU Usage Cost. The total CPU usage cost is the sum of all CPU costs incurred over all time-periods for all servers:

$$\sum_{s_j\in S}\left(\sum_{p_t\in T\wedge o_{jt}=0} Esby_j + \sum_{p_t\in T\wedge o_{jt}=1} \tau_j cpu_{jt} + Emin_j^{'}\right)$$

6 Solution Method: Large Neighbourhood Search

An instance of the energy minimisation problem of data centre as described in the previous section can be very large. As it is an online problem, the challenge is to solve a very large instance in a very limited time. Constraint-based systematic search [12] has shown strong performance, but it does not scale well in terms of time and space for very large instances. Local Search (LS) and Large Neighbourhood Search (LNS) methods have shown remarkable performance on very large instances. A LS method moves from a solution to another by performing a small number of changes (and therefore small improvement) at each iteration, while a LNS method can allow a large number of changes (and possibly large improvement) at each iteration. A meta-heuristic is generally used with LS to escape from local minima, but it is generally unnecessary for LNS. LNS attempts to combine the power of systematic search with the scalability of local search.

In this section we describe a LNS approach for solving the problem formulated in the previous section. The overall solution method is shown in Figure 6. We first find the initial assignment of virtual machines to servers for all time-periods. We maintain a current assignment, which is initialised with the initial solution. At each iteration, we select a subset of the pairs of virtual machines and time-periods to be reassigned and, accordingly, create the sub-problem. We solve the resulting sub-problem with a threshold on the number of failures, and keep the best solution found as our new current assignment. The search stops when the total elapsed time is greater than the given time threshold. Notice that the decision variables can be restricted to the x variables as once an assignment of the virtual machines to the servers is known at all time-periods, the rest of the variables are assigned by propagation and the cost function is fully known.

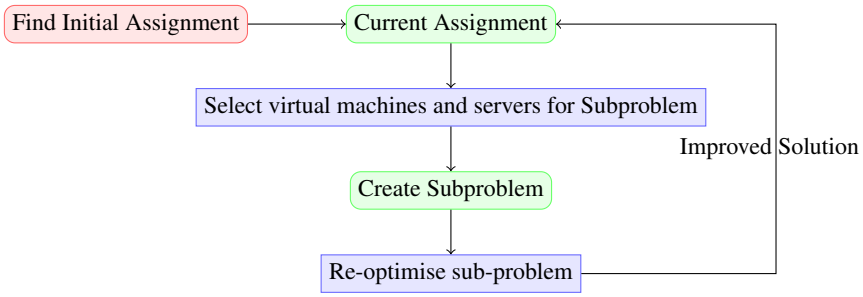

Fig. 6. Principle of the LNS approach.

6.1 Finding Initial Feasible Solution

The pseudo-code for finding an initial feasible solution is depicted in Algorithm 1. The algorithm requires problem $\mathcal{P}$ as input which is composed of Constraints (1)–(9) without any objective function as the task is to find any feasible solution. In the first phase (Lines 3–10) it iterates over a set of unassigned decision variables, denoted by *uvars*, and tries to extend the current partial solution denoted by *sol*. If it succeeds then the current partial solution is updated, otherwise the set of variables, denoted by *fvars*, that failed to find any assignment is updated. In the second phase (Lines 10–21) it first resets the set of unassigned variables to the set of failed variables. For each failed virtual machine a server is selected and, until the assignment of the selected server to the selected virtual machine is consistent, it finds a constraint C that has failed, relaxes the current partial solution by removing a decision variable involved in the failed constraint, and updates the set of unassigned decision variables. The algorithm terminates when all the virtual machines over all time-periods are assigned servers, otherwise it repeatedly executes the first phase followed by the second phase.

6.2 Subproblem Selection

A key observation was that selecting a set of virtual machines from only some servers for reassignment works better than selecting them from many servers. Therefore, we first select a time-period t_b, and then select a number of servers, denoted by k_s, and then from each selected server we select a number of virtual machines that are assigned to them for the time-period t_b. The number of virtual machines that we want to reassign from each selected server is bounded by an integer parameter k_m. Each selected virtual machine that we want to reassign in time-period t_b is also selected from its servers for the time-periods ranging between t_a and t_c such that $t_a \leq t_b \leq t_c$ and $t_c - t_a$ is bounded by an integer parameter k_t. Notice that a virtual machine may not be necessarily running on the same server for all the time-periods between t_a and t_c inclusive. Initially k_s is set to 1, it is incremented as search progresses, and it is re-initialised to 1 when it exceeds 10. Similarly k_t is initially set to 1, it is incremented as search progresses, and it is re-initialised to 1 when it exceeds the maximum number of time-periods h. Depending on

Algorithm 1 findInitialFeasibleSolution($\mathcal{P}$)

```
1:  uvars ← {x_it | v_i ∈ V ∧ p_t ∈ T}; sol ← ∅; fvars ← ∅;
2:  while uvars ≠ ∅ do
3:      while uvars ≠ ∅ do
4:          select & remove any x_it from uvars;
5:          if ∃s_j ∈ A_i s.t. P ∧ sol ∧ (x_it = j) is satisfiable then
6:              sol ← sol ∪ {(x_it = j)}
7:          else
8:              fvars ← fvars ∪ {x_it}
9:          end if
10:     end while
11:     uvars ← fvars
12:     while fvars ≠ ∅ do
13:         select & remove any x_it from fvars;
14:         select any s_j from A_i
15:         while P ∧ sol ∧ (x_it = j) is not satisfaible do
16:             determine any constraint C ∈ P that is not satisfiable
17:             select any x_i't' involved in C such that x_i't' ∈ sol
18:             sol ← sol − {(x_i't' = j')}
19:             uvars ← uvars ∪ {x_i't'}
20:         end while
21:     end while
22: end while
```

the value of k_s and k_t a fixed value of k_m is used. The total number of decision variables selected for reassignment for a sub-problem is bound by $k_s \times k_m \times k_t$.

6.3 Create and Re-optimise Subproblem

The conventional approach for creating a sub-problem would be to reset all the domains of the variables, reassign the servers to the virtual machines (for the appropriate time-periods) which are not chosen for reassignment, and perform constraint propagation before searching the resulting sub-problem. The reason for doing this is that existing solvers are typically designed for systematic backtracking search. However, in LNS one moves from one partial assignment to another in a non-systematic way and unfortunately no support is provided for updating the state of the problem domains incrementally. This way of creating a sub-problem can be a bottleneck for solving very large problems in a very limited time especially if the size of the sub-problem is considerably smaller than the size of the full problem. The reason is that the number of iterations that one would like to perform will increase as the size of the problem increases in which case the time spent in creating the subproblems will also increase. We therefore use the technique described in [10] for replenishing the domains via incremental recomputation. When a set of decisions are undone, the constraints are used explicitly to determine which removed values can be added back to the current domains. The advantage is that it is independent on the order in which the assignments are undone and, therefore, it can be very efficient for creating subproblems.

We use systematic branch and bound search with a threshold on the number of failures for solving a given sub-problem. At each node of the search tree constraint propagation is performed to reduce the search space. We use a random variable ordering heuristic for selecting decision variables. The value ordering heuristic for selecting a server for a given pair of virtual machine and time-period is based on the minimum increment in the objective cost, while ties are broken randomly.

7 Empirical Results

In this section we present empirical results to demonstrate the effectiveness of our large neighbourhood search approach for the constraint optimisation problem as described in Sections 4 and 5.

Approaches. We compare three approaches: the MIP formulation of the problem, the Temporal Greedy approach (TG), and large neighbourhood search (LNS) for the COP model. The detailed presentation of the Mixed Integer linear Programming (MIP) formulation is omitted due to lack of space. The Temporal Greedy (TG) is the currently employed approach in the platform of the industrial partners. It proceeds by decomposing time and is, therefore, more scalable than the MIP approach. It greedily solves the problem period by period using the MIP model restricted to one period (enforcing the known assignment of the previous period). Each time-period is used as a starting period as long as there is time left and, therefore, if required the assignment is extended in both directions towards the beginning and towards the end. In order to compare different upper bounds, we also computed lower bounds (LB) based on column generation with a 2 hour time-limit. The details of the lower bound computation is presented in [4].

Benchmarks. The industry partners provided 74 problem instances, where the maximum number of virtual machines, servers, and time-periods are 242, 20 and 287 respectively. All the instances are available online.[6] We observed on this benchmark that the CPU constraint is the tight one, as opposed to the memory constraint which is always satisfiable. Based on the original instances we also generated larger instances by just duplicating each virtual machine and each server. Out of 74 original instances, 2 instances then became unsatisfiable because of the migration constraints that restrict the movement of virtual machines to different servers over different time-periods. The result is that the increase in the total CPU requirements of the virtual machines running on a server for one or more time-periods exceeds the maximum capacity of the server, and hence the problem becomes unsatisfiable.

Evaluation. The time-limit is 600 seconds unless otherwise stated. If an approach fails to solve an instance within the time-limit then 600 is recorded as its solution time. All experiments were carried out on a Dual Quad Core Xeon CPU, running Linux 2.6.25 x64, with 11.76 GB of RAM, and 2.66 GHz processor speed. The MIP solver used is CPLEX 12.5 with default parameters. For the LNS approach we extended the

[6] `http://www.4c.ucc.ie/~dm6/energetic.tar.gz`

solver used for the machine reassignment problem of ROADEF.[7] All algorithms were implemented in C.

For each problem instance LNS was allowed to run for 600 seconds. Therefore, we report the cpu time (denoted cpu) for only MIP and TG as in some cases they were terminated before the time-limit. For each approach we also report the number of instances (denoted by #nu) for which an approach failed to find a feasible solution. The *gaps* for upper bounds reported by different approaches are computed as $\frac{100\times(ub-lb)}{lb}$ respectively. To compute mean/median/max values of gaps or time of a given approach, we exclude the instances where it fails to return any feasible solution.

Table 1. Summary of results obtained using MIP, LNS and TG approaches with 600 seconds time-limit over 74 original instances.

	LNS	**MIP**		**TG**	
	gap	gap	cpu	gap	cpu
Mean	0.50	0.03	191.92	7.00	42.50
Median	0	0	2.67	0.05	1.45
Max	4.57	0.72	600	119.35	600
#nu	**0**	3		1	

Original Instances. Table 1 gives an overview of the results by reporting over the original 74 instances the average/median/max values of the gap to the best known lower bound, the *cpu* time, and the number of instances, *#nu*, where an approach fails to return any results within the time-limit. Out of 74 instances, MIP is able to find solutions for 71 instances within the time-limit out of which 54 are proved optimal. It thus failed for 3 instances where the space requirement for CPLEX exceeded 11GB. Notice that the largest instance in the original set has 1,389,080 decision variables. Clearly, MIP-based systematic search cannot scale in terms of time and memory. TG is able to find solutions for 73 instances (so it failed on one instance), out of which 26 are optimal. Its quality deteriorates severely when one should anticipate expensive peaks in demand by appropriately placing virtual machines several time-periods before the peak. This can be seen in Table 2 where the maximum gap is 119.35%. LNS succeeds in finding feasible solutions for all instances within 2 seconds, on average, but it was terminated after 600 seconds and for 41 instances it found optimal solutions. Its average gap to the best known lower bound is less than 0.5% showing that LNS scales very well both in quality and problem size. Table 2 also gives the results for a few hard original instances.

Larger Instances. For larger instances MIP failed to find solutions for 7 instances while TG failed to find solutions for 3 instances out of 72 satisfiable instances as shown in Table 3. The increase in the size of the instances has significantly deteriorated the performances of MIP and TG in terms of time and gap when compared to the original set of instances. LNS is the only approach that managed to find solutions for all instances.

[7] http://www.sourceforge.net/projects/machinereassign/

Table 2. Comparison of upper bounds of the various approaches with 600 seconds time-limit on a few specific instances. The first part corresponds to the original instances while the second part corresponds to the generated instances.

			LB		**LNS**	**MIP**		**TG**	
n	m	h	lb	cpu	ub	ub	cpu	ub	cpu
32	3	96	25404.7	14.8	25586.7	**25575.7**	600	36049.7	112.3
36	3	287	126730.1	248.0	**127018.6**	127654.4	600	127036.6	600
242	20	24	38614.2	600	**40362.5**	-	600	43027.6	14.2
242	20	287	431703.9	600	**439926.2**	-	600	-	600
242	20	24	36890.8	56.1	37701.6	-	600	**36897.4**	600
90	7	8	12656.82	0.1	11728.2	**11435.3**	600	11435.5	1.5
64	6	15	7695.6	64.9	**8703.6**	9657.6	600	10233.6	5.74
64	6	96	48169.3	470.3	**53917.3**	-	600	66407.4	84.24
484	40	24	74098.1	600	**86748.8**	-	600	92006.2	63.5
484	40	287	848619.4	1200	**893463.6**	-	600	-	600
136	10	16	15529.3	74.72	15519.3	-	600	**15272.3**	70.52
484	40	24	73781.5	241.32	76240.9	-	600	**73791.44**	600
60	6	15	9857.76	104.82	**11302**	-	600	13407.7	4.18
72	6	287	232222.9	1200	**240679**	565145	600	250104.1	600
108	14	8	9094.86	164.33	**9555.49**	9720.67	600	-	600
72	10	16	18337.4	223.4	**18416.5**	35911.7	600	18556.21	2.57
60	12	24	25484.5	8.29	25535.3	62520.2	600	**25524.59**	5.85
66	6	1	30558.94	0.1	49013.31	**30558.94**	0.1	30558.94	0.1
180	14	8	22864.6	364.42	26427.47	-	600	**22971.3**	32.47

The maximum gap for LNS is for an instance for which both MIP and TG are able to solve it optimally. This instance has only one time-period but the packing part of the problem is harder because of the migration constraints. We note that when the number of time-periods is 1 both MIP and TG are equivalent. As we use a random variable selection heuristic for LNS, it could not perform as well as the systematic and complete search of MIP/TG on that particular instance. Table 2 also gives the results for a few larger instances.

Table 3. Summary of results obtained using MIP, LNS and TG approaches with 600 second time-limit over 72 instances which are generated by duplicating each virtual machine and each server of each original instance.

	LNS	**MIP**		**TG**	
	gap	gap	cpu	gap	cpu
Mean	2.86	10.95	499	9.71	48.15
Median	0.20	0.08	600	0.18	5.59
Max	60.38	145.32	600	120.45	169.16
#nu	**0**	7		3	

Any-time Behavior. Having seen the good performance of LNS we also investigated the impact of different time-limits on LNS. The time-out limit of 600 seconds was defined by our industrial partners. We also solved all the 146 instances with 300 and 150 second time-outs. The results are summarised in Table 4. These results suggest that LNS has a very good any-time behaviour and it can find high quality solutions very quickly. When the time-limit is 150 seconds, it failed to find a solution for only 1 instance which has 5,556,320 decision variables. For this instance it requires at least 200 seconds to find an initial feasible solution and the majority of that effort is spent in the first phase of Algorithm 1, where it tries each choice at least once.

Table 4. Comparison of LNS over 146 instances for 600, 300 and 150 second time-limits.

	600s	300s	150s
Mean Gap	1.669	1.725	1.711
Median Gap	0.089	0.093	0.095
#nu	0	0	1

8 Conclusion

We presented a constraint optimisation formulation of the energy minimisation problem for data centres. We developed a tool that uses constraint programming and large neighbourhood search for solving large problem instances in very limited time. Empirical results on real benchmarks assert that our LNS approach is scalable, thus suited for solving large instances. The presented approach has good anytime behaviour which is important when solutions must be reported subject to a time limit. Currently, we are not taking advantage of multi-cores capabilities that might be available while solving the problem. We plan to explore this opportunty in the future.

References

1. A framework for data center energy productivity. Technical report, The green grid, 2008.
2. Efficience des data centers, les retombés du projet energetic. Technical report, EnergeTIC, www.vesta-system.cades-solutions.com/images/vestalis/4/energetic_white%20paper.pdf, 2013.
3. Eyal Bin, Ofer Biran, Odellia Boni, Erez Hadad, Elliot K Kolodner, Yosef Moatti, and Dean H Lorenz. Guaranteeing high availability goals for virtual machine placement. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 700–709. IEEE, 2011.
4. Hadrien Cambazard, Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. Bin packing with linear usage costs – an application to energy management in data centres. In *The 19th International Conference on Principles and Practice of Constraint Programming*, 2013.
5. Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centres. In *SOSP*, pages 103–116, 2001.

6. Corentin Dupont, G Giuliani, F Hermenier, T Schulze, and A Somov. An energy aware framework for virtual machine placement in cloud federated data centres. In *Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on*, pages 1–10. IEEE, 2012.
7. James Glanz. Power, Pollution and the Internet. `http://www.nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image.html`, September 2012.
8. Fabien Hermenier, Sophie Demassey, and Xavier Lorca. Bin-Repacking Scheduling in Virtualized Datacenters. In *17th International Conference on Principles and Practice of Constraint Programming (CP'11)*, Lecture Notes in Computer Science, Perrugia, Italy, 2011. Springer-Verlag.
9. Heeseok Lee and Taeho Park. Allocating data and workload among multiple servers in a local area network. *Inf. Syst.*, 20(3):261–269, 1995.
10. Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. Comparing solution methods for the machine reassignment problem. In Michela Milano, editor, *CP*, volume 7514 of *Lecture Notes in Computer Science*, pages 782–797. Springer, 2012.
11. Vinicius Petrucci, Orlando Loques, and Daniel Mosse. A dynamic conguration model for power-efficient virtualized server clusters. In *Proceedings of the 11th Brazilian Workshop on Real-Time and Embedded Systems*, 2009.
12. Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Elsevier Science Inc., 2006.